Stochastic Logic Programs*

Stephen Muggleton
Oxford University Computing Laboratory,
Parks Road,
Oxford, OX1 3QD,
United Kingdom.

Abstract

One way to represent a machine learning algorithm's *bias* over the hypothesis and instance space is as a pair of probability distributions. This approach has been taken both within Bayesian learning schemes and the framework of U-learnability. However, it is not obvious how an Inductive Logic Programming (ILP) system should best be provided with a probability distribution. This paper extends the results of a previous paper by the author which introduced *stochastic logic programs* as a means of providing a structured definition of such a probability distribution. Stochastic logic programs are a generalisation of stochastic grammars. A stochastic logic program consists of a set of labelled clauses $p : C$ where p is from the interval $[0, 1]$ and C is a range-restricted definite clause. A stochastic logic program P has a distributional semantics, that is one which assigns a probability distribution to the atoms of each predicate in the Herbrand base of the clauses in P. These probabilities are assigned to atoms according to an SLD-resolution strategy which employs a stochastic selection rule. It is shown that the probabilities can be computed directly for *fail-free* logic programs and by normalisation for arbitrary logic programs. The stochastic proof strategy can be used to provide three distinct functions: 1) a method of sampling from the Herbrand base which can be used to provide selected targets or example sets for ILP experiments, 2) a measure of the information content of examples or hypotheses; this can be used to guide the search in an ILP system and 3) a simple method for conditioning a given stochastic logic program on samples of data. Functions 1) and 3) are used to measure the generality of hypotheses in the ILP system Progol4.2. This supports an implementation of a Bayesian technique for learning from positive examples only.

*This paper is an extension of a paper with the same title which appeared in [12]

1 Introduction

The integration of logic and probability theory has been the subject of many studies [1, 3, 9, 5, 6, 17, 20]. It has also been investigated within logic programming [16, 19]. Recently the motivation for many such studies has been the development of formalisms for rule-based expert systems which employ uncertain reasoning.

By contrast, the motivation for the present paper comes from machine learning. One way to represent a machine learning algorithm's *bias* over the hypothesis and instance space is as a pair of probability distributions. This approach has been taken in various ways within the frameworks of PAC-learnability [21], Bayesian learning [2, 7] and U-learnability [11, 14]. However, it is not obvious how a machine learning algorithm, in particular an Inductive Logic Programming (ILP) system [10, 15] should best be provided with a probability distribution over a set of logical formulae. This paper proposes *stochastic logic programs* (SLPs) as a means of providing a structured definition of such a probability distribution. SLPs are a generalisation of stochastic grammars [8]. Although they have a distributional semantics, SLPs' relationship to Probabilistic Logic Programs [16] and BS-programs [19] is unclear.

hmm?

This paper is organised as follows. In Section 2 the formal framework for U-learnability is introduced. Stochastic grammars are then defined and described in Section 3. Section 4 introduces stochastic logic programs as a generalisation of stochastic grammars. Section 5 describes a Prolog implementation of stochastic logic programs. A discussion of research issues and applications of stochastic logic programs concludes the paper in Section 6.

2 U-learnability

The following is a variant of the U-learnability framework presented in [11, 14]. The teacher starts by choosing distributions $D_{\mathcal{H}}$ and D_X from the family of distributions $\mathcal{D}_{\mathcal{H}}$ and $\mathcal{D}_X$ over concept descriptions $\mathcal{H}$ (wffs with associated bounds for time taken to test entailment) and instances X (ground wffs) respectively. The teacher uses $D_{\mathcal{H}}$ and D_X to carry out an infinite series of teaching sessions. In each session a target theory T is chosen from $D_{\mathcal{H}}$. Each T is used to provide labels from $\{\blacksquare, \square\}$ (True, False) for a set of instances randomly chosen according to distribution D_X. The teacher labels each instance x_i in the series $\langle x_1, .., x_m \rangle$ with $\blacksquare$ if $T \models x_i$ and $\square$ otherwise. An hypothesis $H \in \mathcal{H}$ is said to explain a set of examples E whenever it both entails and is consistent with E. On the basis of the series of labelled instances $\langle e_1, e_2, .., e_m \rangle$, a Turing machine learner L produces a sequence of hypotheses $\langle H_1, H_2, ..H_m \rangle$ such that $H_i \in \mathcal{H}$ explains $\{e_1, .., e_i\}$. H_i must be suggested by L in expected time bounded by a fixed polynomial function of i. The teacher stops a session once the learner suggests hypothesis H_m with expected error less than ϵ for the label of any x_{m+1} chosen

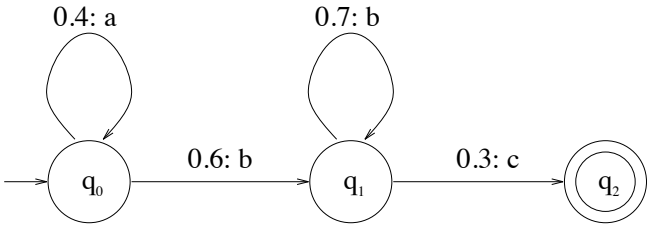

Figure 1: Stochastic automaton.

randomly from D_X. $\langle D_{\mathcal{H}}, D_X \rangle$ is said to be U-learnable if and only if there exists a Turing machine learner L such that for any choice of δ and ϵ $(0 < \delta, \epsilon \leq 1)$ with probability at least $(1 - \delta)$ in any of the sessions m is less than a fixed polynomial function of $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$.

In [14] positive results were given for the U-learnability of time-bounded logic programs. For these proofs it was assumed that a logic program P could be chosen by the teacher according to a distribution $D_{\mathcal{H}}$ and that instances could be chosen from the Herbrand base of P according to a distribution D_X. Clearly both $D_{\mathcal{H}}$ and D_X are functions over countably infinite domains. Stochastic grammars provide one approach to defining a probability distribution over a countably infinite set.

3 Stochastic grammars

3.1 Stochastic automata

Stochastic automata, otherwise called Hidden Markov Models [18], have found many applications in speech recognition. An example is shown in Figure 1. Stochastic automata are defined by a 5-tuple $A = \langle Q, \Sigma, q_0, F, \delta \rangle$. Q is a set of states. Σ is an alphabet of symbols. q_0 is the initial state and $F \subseteq Q$ ($F = \{q_2\}$ in Figure 1) is the set of final states. $\delta : (Q \setminus F) \times \Sigma \rightarrow Q \times [0, 1]$ is a stochastic transition function which associates probabilities with labelled transitions between states. The sum of probabilities associated with transitions from any state $q \in (Q \setminus F)$ is 1.

In the following λ represents the empty string. The transition function $\delta^* : (Q \setminus F) \times \Sigma^* \rightarrow Q \times [0, 1]$ is defined as follows. $\delta^*(q, \lambda) = \langle q, 1 \rangle$. $\delta^*(q, au) = \langle q_{au}, p_a p_u \rangle$ if and only if $\delta(q, a) = \langle q_a, p_a \rangle$ and $\delta^*(q_a, u) = \langle q_{au}, p_u \rangle$. The probability of u being accepted from state q in A is defined as follows. $Pr(u|q, A) = p$ if $\delta^*(q, u) = \langle q', p \rangle$ and $q' \in F$. $Pr(u|q, A) = 0$ otherwise.

Theorem 1 Probability of a string being accepted from a particular state. *Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a stochastic automaton. For any $q \in Q$ the*

following holds.

$$\sum_{u \in \Sigma^*} Pr(u|q, A) = 1.$$

Proof. *Suppose the theorem is false. Either $q \in F$ or $q \notin F$. Suppose $q \in F$. Then by the definition of stochastic automata q has no outgoing transitions. Therefore by definition $Pr(u|q, A)$ is 1 for $u = \lambda$ and 0 otherwise, which is in accordance with the theorem. Therefore suppose $q \notin F$. Suppose in state q the transitions are $\delta(q, a_1) = \langle q, p_1 \rangle, \ldots, \delta(q, a_n) = \langle q, p_n \rangle$. Then each string u is accepted in proportions $p_1, \ldots, p_n$. according to its first symbol. That is to say, $\sum_{u \in \Sigma^*} Pr(u|q, A) = p_1 + .. + p_n$. But according to the definition of δ, $p_1 + \ldots + p_n = 1$, which means $\sum_{u \in \Sigma^*} Pr(u|q, A) = 1$. This contradicts the assumption and completes the proof.* □

If the probability of u being accepted by A is now defined as $Pr(u|A) = Pr(u|q_0, A)$ then the following corollary shows that A defines a probability distribution over Σ^*.

Corollary 2 Stochastic automata represent probability distributions. *Given stochastic automaton A.*

$$\sum_{u \in \Sigma^*} Pr(u|A) = 1.$$

Proof. *Special case of Theorem 1 when $q = q_0$.* □

The following example illustrates the calculation of probabilities of strings.

Example 3 Probabilities associated with strings. *For the automaton A in Figure 1 we have $Pr(abbc|A) = 0.4 \times 0.6 \times 0.7 \times 0.3 = 0.0504$. $Pr(abac|A) = 0$.*

A can also be viewed as expressing a probability distribution over the language $L(A) = \{u : \delta^*(q_0, u) = \langle q, p \rangle \text{ and } q \in F\}$. The following theorem places bounds on the probability of individual strings in $L(A)$. The notation $|u|$ is used to express the length of string u.

Theorem 4 Probability bounds. *Let $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ be a stochastic automaton and let p_{min}, p_{max} be respectively the minimum and maximum probabilities of any transition in A. Let $u \in L(A)$ be a string.*

$$p_{min}^{|u|} \leq Pr(u|A) \leq p_{max}^{|u|}.$$

Proof. *$Pr(u|A) = \prod_{i=1}^{|u|} p_i$, where p_i is the probability associated with the ith transition in A accepting u. Clearly each p_i is bounded below by p_{min} and above by p_{max}, and thus $p_{min}^{|u|} \leq Pr(u|A) \leq p_{max}^{|u|}$.* □

This theorem shows that a) all strings in $L(A)$ have non-zero probability and b) stochastic automata express probability distributions that decrease exponentially in the length of strings in $L(A)$.

$$0.4 : q_0 \rightarrow aq_0$$
$$0.6 : q_0 \rightarrow bq_1$$

$$0.7 : q_1 \rightarrow bq_1$$
$$0.3 : q_1 \rightarrow cq_2$$

$$1.0 : q_2 \rightarrow \lambda$$

Figure 2: Labelled production rule representation of stochastic automaton.

3.2 Labelled productions

Stochastic automata can be equivalently represented as a set of labelled production rules. Each state in the automaton is represented by a non-terminal symbol and each δ transition $\langle q, a\rangle \rightarrow \langle q', p\rangle$ is represented by a production rule of the form $p : q \rightarrow aq'$. Figure 2 is the set of labelled production rules corresponding to the stochastic automaton of Figure 1. Strings can now be generated from this stochastic grammar by starting with the string q_0 and progressively choosing productions to rewrite the leftmost non-terminal randomly in proportion to their probability labels. The process terminates once the string contains no non-terminals. The probability of the generated string is the product of the labels of rewrite rules used.

3.3 Stochastic context-free grammars

Stochastic context-free grammars [8] can be treated in the same way as the labelled productions of the last section. However, the following differences exist between the regular and context-free cases.

- To allow for the expression of context-free grammars the left-hand sides of the production rules are allowed to consist of arbitrary strings of terminals and non-terminals.
- Since context-free grammars can have more than one derivation of a particular string u, the probability of u is the sum of the probabilities of the individual derivations of u.
- The analogue of Theorem 4 holds only in relation to the length of the derivation, not the length of the generated string.

Example 5 The language $a^n b^n$. *Figure 3 shows a stochastic context-free grammar G expressed over the language $a^n b^n$. The probabilities of generated strings are as follows.* $Pr(\lambda|G) = 0.5$, $Pr(ab|G) = 0.25$, $Pr(aabb|G) = 0.125$.

$$0.5 : S \rightarrow \lambda$$
$$0.5 : S \rightarrow aSb$$

Figure 3: Stochastic context free grammar

$$0.5 : coin(0) \leftarrow$$
$$0.5 : coin(1) \leftarrow$$

Figure 4: Simple SLP

4 Stochastic logic programs

Every context-free grammar can be expressed as a definite clause grammar [4]. For this reason the generalisation of stochastic context-free grammars to stochastic logic programs (SLPs) is reasonably straightforward. First, a definite clause C is defined in the standard way as having the following form.

$$A \leftarrow B_1, \ldots, B_n$$

where the atom A is the head of the clause and $B_1, \ldots, B_n$ is the body of the clause. C is said to be range-restricted if and only if every variable in the head of C is found in the body of C. A *stochastic clause* is a pair $p : C$ where p is in the interval $[0, 1]$ and C is a range-restricted clause. A set of stochastic clauses P is called a *stochastic logic program* if and only if for each predicate symbol q in P the probability labels for all clauses with q in the head sum to 1.

Example 6 Coin example. *Figure 4 shows a simple SLP which mimics the action of a fair coin. The probability of the coin coming up either head-side up* (0) *or tail-side up* (1) *is* 0.5.

4.1 Stochastic SLD-refutations

For SLPs the stochastic refutation of a goal is analogous to the stochastic generation of a string from a set of labelled production rules. Suppose that P is an SLP. Then $n(P)$ will be used to express the logic program formed by dropping all the probability labels from clauses in P. A stochastic SLD procedure will be used to define a probability distribution over the Herbrand base of $n(P)$. The stochastic SLD-derivation of atom a is as follows. Suppose $\leftarrow g$ is a unit goal with the same predicate symbol as a, no function symbols and distinct variables. Next suppose that there exists an SLD-refutation of $\leftarrow g$ with answer substitution θ such that $g\theta = a$. Since all clauses in $n(P)$ are range-restricted, θ is necessarily a ground substitution. The probability of each clause selection

in the refutation is as follows. Suppose the first atom in the subgoal $\leftarrow g'$ can unify with the heads of stochastic clauses $p_1 : C_1, \ldots, p_n : C_n$, and stochastic clause $p_i : C_i$ is chosen in the refutation. Then the probability of this choice is $\frac{p_i}{p_1+\ldots+p_n}$. The probability of the derivation of a is the product of the probability of the choices in the refutation. As with stochastic context-free grammars, the probability of a is then the sum of the probabilities of the derivations of a.

This stochastic SLD-strategy corresponds to a distributional semantics [19] for P. That is, each atom a in the success set of $n(P)$ is assigned a non-zero probability (due to the completeness of SLD-derivation). For each predicate symbol q the probabilities of atoms in the success set of $n(P)$ corresponding to q sum to 1 (the proof of this is analogous to Theorem 1).

4.2 Fail-free SLPs

It should be noted that the definition of SLPs in the previous section has problems. For instance, consider the case in which P has an empty success set. In this case the probability distribution is not well defined since it does not sum to 1. A less extreme case occurs when at least some derivations exist, though other derivations reach a deadend in which the goal $\leftarrow g'$ cannot unify with the heads of any clauses. In this case the probabilities of individual atoms must be normalised by multiplying each derivation probability by the reciprocal of the sum of all such probabilities.

Fail-free logic programs avoid this issue, since no selection choice leads to enforced backtracking. Fail-free clauses, logic programs and SLPs are defined as follows[1].

Definition 7 Fail-free clause. *A clause $h \leftarrow B$ is said to be fail-free if and only if B contains no function symbols and each variable occurs at most once in B.*

Definition 8 Fail-free logic programs. *A logic program P is fail-free if and only if each clause in P is definite and fail-free, and each predicate symbol in the body of each clause in P occurs in the head of at least one clause in P.*

Definition 9 Fail-free stochastic logic programs. *A stochastic logic program P is fail-free if and only if $n(P)$ is a fail-free logic program.*

We now show fail-free logic programs avoid forced backtracking.

Theorem 10 Non-backtracking of fail-free logic programs. *Let P be a fail-free logic program and $\leftarrow g$ be a fail-free goal containing only predicate symbols found in P. Irrespective of the choices made in the stchastic SLD-derivation of goal $\leftarrow g$ there will be no subgoal $\leftarrow g'$ which fails to resolve with*

[1] Although the following constraints are sufficient to avoid forced backtracking, it is not clear whether they are also necessary for definition of this class.

$$0.5 : nate(0) \leftarrow$$
$$0.5 : nate(s(N)) \leftarrow nate(N)$$

Figure 5: Exponential distribution over natural numbers

the definition of any predicate in g'.

Proof. *Assume the theorem is false. First note that the resolvent of any fail-free goal and a fail-free definite clause is itself fail-free and that the unification involved is one-sided. However, there must exist an intermediate goal $\leftarrow g'$ with substitution θ' which fails to resolve with the predicate definition for a predicate p in P. But since every predicate symbol in $\leftarrow g$ has a definition in P and each predicate symbol in the body of each clause in P has a definition in P it must be that $\leftarrow g'\theta'$ fails to unify with any of the clauses in the definition of p. However, since all unifications are one-sided, θ' will contain no substitutions for the variables in $\leftarrow g'$, and thus the first atom of $\leftarrow g'\theta'$ will have no function symbols and distinct variables, and thus must be able to resolve with all clauses of the corresponding definition. This contradicts the assumption and completes the proof.* □

Note that the class of fail-free logic programs includes all normal unary definitions of types (such as *list/1* or *natural/1*) as well as the standard recursive definitions of predicates such as *member/2* and *append/3*.

4.3 Polynomial distributions

It is reasonable to ask whether Theorem 4 extends in some form to SLPs. The distributions described in [14] include both those that decay exponentially over the length of formulae and those that decay polynomially. SLPs can easily be used to describe an exponential decay distribution over the natural numbers as follows.

Example 11 Exponential distribution. *Figure 5 shows a recursive SLP P which describes an exponential distribution over the natural numbers expressed in Peano arithmetic form. The probabilities of atoms are as follows. $Pr(nate(0)|P) = 0.5$, $Pr(nate(s(0))|P) = 0.25$ and $Pr(nate(s(s(0)))|P) = 0.125$. In general $Pr(nate(N)|P) = 2^{-N-1}$.*

However, SLPs can also be used to define a polynomially decaying distribution over the natural numbers as follows.

Example 12 Polynomial distribution. *Figure 6 shows a recursive SLP P which describes a polynomial distribution over the natural numbers expressed in reverse binary form. Numbers are constructed by first choosing the length of*

$$1.0 : natp(N) \leftarrow nate(U), bin(U, N)$$

$$0.5 : bin(0, [1]) \leftarrow$$
$$0.5 : bin(s(U), [C|N]) \leftarrow coin(C), bin(U, N)$$

Figure 6: Polynomial distribution over natural numbers

the binary representation and then filling out the binary expression by repeated tossing of a fair coin (see Figure 4). Since the probability of choosing a number N of length $log_2(N)$ is roughly $2^{-log_2(N)}$ and there are $2^{log_2(N)}$ such numbers, each with equal probability, $Pr(natp(N)|P) \approx 2^{-2log_2(N)} = N^{-2}$.

5 A Prolog implementation

Stochastic logic programs can be used to provide three distinct functions.

1. **A sampler.** A method of sampling from the Herbrand base which can be used to provide selected targets or example sets for ILP experiments, Sampling targets requires an SLP that implements a grammar describing the hypothesis space.

2. **Information content.** A measure of the information content of examples or hypotheses. The information content of atom a relative to SLP P is taken here as simply $-log_2(Pr(a|P))$. This could be used to help guide the search in an ILP system.

3. **A conditioner.** A simple method for conditioning a given stochastic logic program on samples of data.

Prolog implementations of these three functions are described in the following sections.

5.1 The predicate *sample/1*

This section describes a Prolog interpreter for SLPs. Figure 7 shows the code for the top-level interpreter, which is similar to a standard 'proves' interpreter for Prolog. Note that backtracking is disabled by the use of cuts. The reason for this is that each atom should be sampled independently of its predecessors. The third clause uses the predicate *clause/3*, the third argument of which is a unique number associated with the returned clause. Though this predicate is non-standard it is relatively straightforward to implement. (It is implemented as a primitive in CProgol4.2 which can be obtained by anonymous ftp from ftp.comlab.ox.ac.uk in directory pub/Packages/ILP/progol4.2).

```
sample((Goal1,Goal2)) :-
        !, sample(Goal1), sample(Goal2). % Conjunction
sample(Goal) :-
        not(clause(Goal,_)), !, Goal.    % System predicate
sample(Head) :-
        bagof([Head,Body,N],clause(Head,Body,N),Bag),
        random_clause(Head,Body,Bag),    % Random choice
        !, sample(Body).                 % User predicate
```

Figure 7: The predicate sample/1

```
random_clause(Head,Body,Bag) :-
        Rand is random,                % 0-1 random number
        choose(Bag,Head,Body,0,Rand,Sum).

choose([],_,_,_,_,0).
choose([[Head,Body,N]|Bag],Head1,Body1,SoFar,Rand,Rest) :-
        label(N,P), SoFar1 is SoFar+P,
        choose(Bag,Head1,Body1,SoFar1,Rand,Rest1),
        Rest is Rest1+P,
        ((var(Body1), P1 is SoFar/(SoFar+Rest), Rand>=P1,
                Head1=Head, Body1=Body);
        true).
```

Figure 8: The predicates random_clause/3 and choose/6

The predicate *random_clause/3* and its sub-predicate *choose/6* are shown in Figure 8. In *choose/6* the probability label of the clause is extracted using the predicate *label/2*, which is also implemented as a primitive in CProgol4.2 (see above). Note that since *P1* is simply a ratio, it is immaterial whether the labels are themselves in the interval $[0, 1]$ or simply arbitrary positive reals. CProgol4.2 by default assigns all clauses a label value of 1. CProgol4.2 also contains alterations to a standard Prolog interpreter which allow efficient sampling of stochastic logic programs using the built-in predicate *sample/3*.

5.2 Information content

Predicates for computing the information content of atoms are shown in Figures 9 and 10. For simplicity it is assumed here that each atom has at most one proof, and that each unification is deterministically chosen given the substitution so far. The predicate *info/3* again acts like a 'proves' interpreter which computes the

```
info(Goal,Bits) :-
        functor(Goal,F,N),
        functor(GGoal,F,N),
        info(GGoal,Goal,Bits1),
        Bits is Bits1.

info((GGoal1,GGoal2),(SGoal1,SGoal2),Bits1+Bits2) :- !,
        info(GGoal1,SGoal1,Bits1),
        info(GGoal2,SGoal2,Bits2).  % Conjunction
info(GHead,SHead,Bits1+Bits2) :-
        bagof([GBody,GN],clause(GHead,GBody,GN),GBag),
        clause(SHead,SBody,SN),
        info_choice(GBag,SN,GBody,0,_,Bits1),
        !, info(GBody,SBody,Bits2). % User predicate
info(Goal,Goal,0) :-
        not(clause(Goal,_)), Goal.  % System predicate
```

Figure 9: The predicates *info/2* and *info/3*

```
info_choice([],_,_,_,0,_).
info_choice([[Body,N]|T],N,Body,SoFar,Rest,Bits) :-
        !, info_choice(T,N,_,SoFar,Rest,_),
        label(N,P), Bits is -log(P/(SoFar+P+Rest))/log(2).
info_choice([[_,N]|T],M,Body,SoFar,Rest+P,Bits) :-
        label(N,P), info_choice(T,M,Body,SoFar+P,Rest,Bits).
```

Figure 10: The predicate *info_choice/6*

probabilities of the choices of a given ground goal (Goal) relative to a general form of the same goal (GGoal). The predicate *info_choice/6* acts much like *choose/6* in Figure 8 except that it computes the negative log probability of the choice.

5.3 The predicate *condition/1*

Predicate *condition/1* in Figure 11 uses *label/1* to condition an SLP according to a given ground goal. The built-in predicate *label/1* in CProgol4.2 simply increments an integer label associated with each clause.

The conditioner can be used to learn the parameters of a given distribution from a given set of ground atomic clauses.

```
condition((Goal1,Goal2)) :-
        !, condition(Goal1), condition(Goal2). % Conjunction
condition(Goal) :-
        not(clause(Goal,_)), !, Goal.          % System predicate
condition(Head) :-
        clause(Head,Body,N), label(N),
        !, condition(Body).                    % User predicate
```

Figure 11: Predicate *condition/1*

6 Discussion

This paper introduces stochastic logic programs as a structural description of a learning system's biases over the hypothesis and instance spaces. Since SLPs seem a simple and natural extension of logic programs it is hoped that they might find further applications within logic programming. Possible areas for application include robotics, planning and natural language.

SLPs have been applied in the problem of learning from positive examples only [13]. This required the implementation of the following function which defines the generality of an hypothesis.

$$g(H) = \sum_{x \in H} D_X(x).$$

The generality is thus the sum of the probability of all instances of hypothesis H. Clearly such a sum can be infinite. However, if a large enough sample is generated from D_X (implemented as an SLP) then the proportion of the sample entailed by H gives a good approximation of $g(H)$. CProgol4.2 (see Section 5.1) uses an implementation of SLPs in this way for learning from positive data.

The implementations of information content and conditioning (Sections 5.2 and 5.3) are both unsatisfactory in that they assume there is only one derivation of every atom. A complete implementation would sum over all derivations. Unfortunately, it is possible that an infinite set of derivations exist for certain atoms. However, since the probabilities associated with such derivations decrease exponentially in the length of the description it may be the case that summing over the short derivations gives good bounds for the complete sum. This question requires further attention.

The author believes that stochastic logic programs provide an important new representation for machine learning and logic programming.

Acknowledgements

Thanks are due to David Haussler of the University of Santa Cruz for providing

me with references to the literature of stochastic grammars. Thanks also for useful discussions on the topics in this paper with Donald Michie, John McCarthy, David Page and Ashwin Srinivasan. This work was supported partly by the Esprit Basic Resaerch Action ILP (6020), the Esprit Long-term Research Action ILP II (LTR 20237), EPSRC grant GR/J46623 on Experimental Application and Development of ILP, EPSRC grant GR/K57985 on Experiments with Distribution-Based Machine Learning and an EPSRC Advanced Research Fellowship held by the author. The author is also supported by a Research Fellowship at Wolfson College Oxford.

References

[1] G. Boole. *The Laws of Thought.* MacMillan & Co., London, 1854.

[2] W. Buntine. *A Theory of Learning Classification Rules.* PhD thesis, School of Computing Science, University of Technology, Sydney, 1990.

[3] R. Carnap. *The logical foundations of probability.* University of Chicago Press, Chicago, 1962.

[4] W.F. Clocksin and C.S. Mellish. *Programming in Prolog.* Springer-Verlag, Berlin, 1981.

[5] A.P. Dempster. A generalisation of bayesian inference. *Journal of the Royal Statistical Society, Series B*, 30:205–247, 1968.

[6] R. Fagin and J. Halpern. Uncertainty, belief and probability. In *Proceedings of IJCAI-89*, San Mateo, CA, 1988. Morgan Kauffman.

[7] D. Haussler, M Kearns, and R. Shapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. In *COLT-91: Proceedings of the 4th Annual Workshop on Computational Learning Theory*, pages 61–74, San Mateo, CA, 1991. Morgan Kauffmann.

[8] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[9] J. Lukasiewicz. Logical foundations of probability theory. In L. Berkowski, editor, *Selected works of Jan Lukasiewicz.* North Holland, Amsterdam, 1970.

[10] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[11] S. Muggleton. Bayesian inductive logic programming. In W. Cohen and H. Hirsh, editors, *Proceedings of the Eleventh International Machine Learning Conference*, pages 371–379, San Mateo, CA, 1994. Morgan-Kaufmann.

[12] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press/Ohmsha, 1995.

[13] S. Muggleton. Learning from positive data. In *Proceedings of the Sixth Inductive Logic Programming Workshop*, Stockholm University, 1996.

[14] S. Muggleton and C.D. Page. A learnability model for universal representations. Technical Report PRG-TR-3-94, Oxford University Computing Laboratory, Oxford, 1994.

[15] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[16] R. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[17] N. Nilsson. Probabilistic logic. *Artificial Intelligence Journal*, 28:71–87, 1986.

[18] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[19] T. Sato. A statistical learning method for logic programs with distributional semantics. In L. Sterling, editor, *Proceedings of the Twelth International conference on logic programming*, pages 715–729, Cambridge, Massachusetts, 1995. MIT Press.

[20] G. Shafer. *A mathematical theory of evidence*. Princeton University Press, Princeton, NJ, 1976.

[21] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.